2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

Data Encoding for Byzantine-Resilient Distributed Gradient Descent

Deepesh Data
University of California, Los Angeles
CA 90095, USA
Email: deepeshdata@ucla.edu

Linqi Song
City University of Hong Kong
Hong Kong SAR
Email: linqi.song@cityu.edu.hk

Suhas Diggavi
University of California, Los Angeles
CA 90095, USA
Email: suhasdiggavi@ucla.edu

***Abstract*—We consider distributed gradient computation, where both data and computation are distributed among m worker machines, t of which can be Byzantine adversaries, and a designated (master) node computes the model/parameter vector, iteratively using gradient descent (GD). The Byzantine adversaries can (collaboratively) deviate arbitrarily from their gradient computation. To solve this, we propose a method based on data encoding and (real) error correction to combat the adversarial behavior. We can tolerate up to $t \leq \lfloor \frac{m-1}{2} \rfloor$ corrupt worker nodes, which is information-theoretically optimal. Our method does not assume any probability distribution on the data. We develop a sparse encoding scheme which enables computationally efficient data encoding. We demonstrate a trade-off between the number of adversaries tolerated and the resource requirement (storage and computational complexity). As an example, our scheme incurs a constant overhead (storage and computational complexity) over that required by the distributed GD algorithm, without adversaries, for $t \leq \frac{m}{3}$.**

I. Introduction

Map-reduce architecture [1] is implemented in many distributed learning tasks, where there is one designated machine (called the master) that computes the model iteratively, based on the inputs from the worker machines, at each iteration, typically using descent techniques, like gradient descent, the Newton's method, etc. The worker nodes perform the required computations using local data, distributed to the nodes [2]. Several other architectures, including having no hierarchy among the nodes have been explored [3].

In several applications of distributed learning, including the Internet of Battlefield Things (IoBT) [4], federated optimization [5], the recruited worker nodes might be partially trusted with their computation. Therefore, an important question is whether can reliably perform distributed computation, taking advantage of partially trusted worker nodes. These (Byzantine) adversaries, can arbitrarily deviate from their specified programs. The problem of distributed computation with Byzantine adversaries has a long history [6], and there has been recent interest in applying this computational model to large-scale distributed learning [7]–[9].

In this paper, we propose a Byzantine-resilient distributed optimization algorithm based on data encoding and error correction (over real numbers). Our proposed algorithm differs from existing Byzantine-resilient distributed learning algorithms in one or more of the following aspects: (i) it does not make statistical assumptions on the data or Byzantine attack patterns; (ii) it is information-theoretically optimal and can tolerate up to a constant fraction ($< 1/2$) of the worker nodes being Byzantine; (iii) it enables a trade-off (in terms of storage and computation overhead in worker nodes) with Byzantine adversary tolerance, without compromising the efficiency at the master node.

Our algorithm encodes the data used by the m worker nodes, using ideas from real-error correction to enable tolerance to Byzantine workers. It develops an efficient "decoding" scheme at the master node, to process the inputs from the workers, to compute the true gradient. It uses a two-phase approach at each iteration of the gradient calculation. Each worker node performs computation *oblivious* to the encoded data, *i.e.,* they perform the same operations as they would have for the original (uncoded) data. The main result (summarized in Theorem 1) demonstrates a trade-off between the Byzantine resilience (in terms of the number of adversarial nodes) and the resource requirement (storage and computational complexity). We can also handle streaming data, where data arrives in batches, rather than being available at the beginning of the computation; see Section V. Finally, the scheme can handle both Byzantine attacks and missing updates (*e.g.,* caused by delay and asynchrony of worker nodes). Though data encoding is a one-time process, it has to be efficient to harness the advantage of distributed computation. We design a sparse encoding process, based on real-error correction [10], which enables efficient encoding, and the worker nodes alternatively locally encode using the sparse structure. This allows encoding with storage redundancy of $2m/(m-2t)$ (which is constant *even* if t is a constant fraction of m), and one-time computational cost for encoding is $O((1+2t)nd)$.

Paper organization. Our problem formulation and the main result are stated in Section II, which includes related work. We present our scheme in Section III, and analyze its use of storage and computational resources in Section IV. In Section V, we extend our encoding procedure to the streaming data model. We conclude with a short discussion in Section VI.

Notation. We denote vectors by bold small letters (e.g., $\mathbf{x}, \mathbf{y}, \mathbf{z}$, etc.) and matrices by bold capital letters (e.g., $\mathbf{A}, \mathbf{F}, \mathbf{S}, \mathbf{X}$, etc.). We denote the size of a matrix $\mathbf{X}$ by

978-1-5386-6596-1/18/$31.00 ©2018 IEEE

$|\mathbf{X}|$. For any positive integer $n \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. For $n_1, n_2 \in \mathbb{N}$, where $n_1 \leq n_2$, we write $[n_1 : n_2]$ to denote the set $\{n_1, n_1 + 1, \ldots, n_2\}$. For any vector $\mathbf{u} \in \mathbb{R}^n$ and any set $\mathcal{S} \subset [n]$, we write $\mathbf{u}_\mathcal{S}$ to denote the $|\mathcal{S}|$-length vector, which is the restriction of $\mathbf{u}$ to the coordinates in the set $\mathcal{S}$. The support of a vector $\mathbf{u} \in \mathbb{R}^n$ is defined by $\mathsf{supp}(\mathbf{u}) := \{i \in [n] : u_i \neq 0\}$. We say that a vector $\mathbf{u} \in \mathbb{R}^n$ is t-sparse if $|\mathsf{supp}(\mathbf{u})| \leq t$.

II. Problem Setting and Our Results

Given a dataset consisting of n data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where, for every data point $(\mathbf{x}_i, y_i)$, $\mathbf{x}_i \in \mathbb{R}^d$ is called the feature vector and $y_i \in \mathbb{R}$ is its ground truth label. We want to learn a *linear regression* model $\mathbf{w} \in \mathbb{R}^d$ that minimizes the squared loss function $J(\mathbf{w})$:

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2}_{J(\mathbf{w})} = \frac{1}{2}\sum_{i=1}^{m} \underbrace{|\mathbf{X}_i\mathbf{w} - \mathbf{y}_i\|^2}_{J_i(\mathbf{w})}, \quad (1)$$

where $(\mathbf{X}_i, \mathbf{y}_i)$ is stored on machine i. This problem is often solved using distributed *Gradient Descent* (GD), where at each iteration t, the workers calculate the local gradient of the cost function $J_i(\cdot)$ at $\mathbf{w}_t$:

$$\nabla J_i(\mathbf{w}_t) = \mathbf{X}_i^T(\mathbf{X}_i\mathbf{w}_t - \mathbf{y}). \quad (2)$$

The master node aggregates the local gradients to compute

$$\nabla J(\mathbf{w}_t) = \mathbf{X}^T(\mathbf{X}\mathbf{w}_t - \mathbf{y}) = \sum_{i=1}^{m} \nabla J_i(\mathbf{w}_t), \quad (3)$$

and then update the linear regression model parameter using $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \nabla J(\mathbf{w}_t)$, where α is the step size or the learning rate, determining the convergence behavior. There are standard choices for it; see for example [11, Chapter 9].

Adversary model. The adversary can corrupt *any* t of the worker nodes[1] and the compromised nodes can send local outcomes that are arbitrarily far away from the actual local outcomes. The adversarial nodes can collude, and can even know the data of other workers. The master node does not know which t worker nodes are corrupted, but knows t, the maximum possible number of adversarial nodes.

Remark 1. *A well-studied problem is that of asynchronous distributed optimization, where the workers can have different delays in updates [12]. One mechanism to deal with this is to wait for a subset of responses, before proceeding to the next iteration, treating the others as missing (or erasures) [13]. Byzantine attacks are quite distinct from such erasures, as the adversary can report wrong local gradients, requiring the master node to create mechanisms to overcome such attacks. If the master node simply aggregates the collected updates as in* (3)*, the computed gradient could be arbitrarily far away from the true one, even with a single adversary [14].*

[1]Our results also apply to a slightly *different* adversarial model, where the adversary can adaptively *choose* which of the t worker nodes to attack at each iteration. However, in this model, the adversary cannot modify the local stored data of the attacked node, as otherwise, over time, it can corrupt all the data, making any defense impossible.

A. Our Approach

Since $\nabla J(\mathbf{w}) = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$, a natural approach for computing the gradient is to compute it in two rounds: (i) compute $\mathbf{u} := \mathbf{X}\mathbf{w} - \mathbf{y}$ in the first round by multiplying $\mathbf{A}^{(1)} := [\mathbf{X} \quad -\mathbf{y}]$ with $[\mathbf{w}^T \ 1]^T$; and (ii) compute $\nabla J(\mathbf{w}) = \mathbf{X}^T\mathbf{u}$ in the second round by multiplying $\mathbf{A}^{(2)} := \mathbf{X}^T$ with $\mathbf{u}$. To combat against the adversarial worker nodes, we do both of these steps using data encoding and error correction (over $\mathbb{R}$); see Figure 1 for a pictorial description of our approach. More specifically, for the first round, we encode $\mathbf{A}^{(1)}$ using a sparse encoding matrix $\mathbf{S}^{(1)} = [(\mathbf{S}_1^{(1)})^T, \ldots, (\mathbf{S}_m^{(1)})^T]^T$ and store $\mathbf{S}_i^{(1)}\mathbf{A}^{(1)}$ at the i'th worker node; and, for the second round, we encode $\mathbf{A}^{(2)}$ using another sparse encoding matrix $\mathbf{S}^{(2)} = [(\mathbf{S}_1^{(2)})^T, \ldots, (\mathbf{S}_m^{(2)})^T]^T$, and store $\mathbf{S}_i^{(2)}\mathbf{A}^{(2)}$ at the i'th worker node. Now, in the 1st round of the gradient computation at $\mathbf{w}$, the master node broadcasts $\mathbf{v} = [\mathbf{w}^T \ 1]^T$ and the i'th worker node replies with $\mathbf{S}_i^{(1)}\mathbf{A}^{(1)}\mathbf{v}$ (a corrupt worker may report an arbitrary vector); upon receiving all the vectors, the master node applies error-correction procedure to recover $\mathbf{u} = \mathbf{A}^{(1)}\mathbf{v}$, which it broadcasts in the 2nd round, and similarly can recover $\mathbf{A}^{(2)}\mathbf{u}$ (which is equal to the gradient) at the end of the 2nd round. Our main result for the Byzantine-resilient distributed GD is as follows:

Theorem 1 (Main Result)**.** *Let $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{n\times(d+1)}$ denote the data. Let m denote the total number of worker nodes. We can compute the gradient exactly in a distributed manner in the presence of t corrupt worker nodes and s stragglers, with the following guarantees, where $\epsilon > 0$ is a free parameter.*

- $(s + t) \leq \left\lfloor \frac{\epsilon}{1+\epsilon} \cdot \frac{m}{2} \right\rfloor$.
- *Total storage requirement is roughly* $2(1 + \epsilon)|\mathbf{X}|$.
- *Computational complexity for each gradient computation:*
  - *At each worker node is* $O((1 + \epsilon)\frac{nd}{m})$.
  - *At the master node is* $O((1 + \epsilon)(n + d)m)$.
- *Total encoding time is* $O\left(nd\left(\frac{\epsilon}{1+\epsilon}m + 1\right)\right)$.

Remark 2. *The statement of Theorem 1 allows for any s and t, as long as $(s + t) \leq \left\lfloor \frac{\epsilon}{1+\epsilon} \cdot \frac{m}{2} \right\rfloor$. As we are handling both erasures and errors in the same way*[2] *the corruption threshold does not have to handle s and t separately. To simplify the discussion, for the rest of the paper, we consider Byzantine corruption, and denote the corrupted set by $\mathcal{I} \subset [m]$ with $|\mathcal{I}| \leq t$, with the understanding that this can also work with stragglers.*

Remark 3. *Let m be an even number. Note that we can get the corruption threshold t to be any number less than $m/2$, but at the expense of increased storage and computation. For any $\delta > 0$, if we want to get δ close to m/2, i.e., $t = m/2 - \delta$, then we must have $(1 + \epsilon) \geq m/2\delta$. In*

[2]When there are *only stragglers*, one can design an encoding scheme where *both* the master and the worker nodes operate oblivious to encoding, while solving a slightly altered optimization problem [13], in which gradients are computed *approximately*, leading to more efficient straggler-tolerant GD.

particular, at $\epsilon = 2$, we can tolerate up to $m/3$ corrupt nodes, with constant overhead in the total storage as well as on the computational complexity. Our encoding is also efficient and requires $O\left(nd\left(\frac{\epsilon}{1+\epsilon}m+1\right)\right)$ time. Note that $O(nd)$ is equal to the time required for distributing the matrix $\mathbf{A}$ among m workers (for performing distributed MV multiplication without *the adversary); and the encoding time in our scheme (which results in an encoded matrix that provides Byzantine-resiliency) a factor of $(2t+1)$ more.*

Remark 4. *On comparing the resource requirements of our method with the plain distributed GD with* no adversarial protection*, we have that, in our scheme (i) the total storage requirement is $O(1+\epsilon)$ factor more (which is just a constant overhead); (ii) the amount of computation at each worker node is $O(1+\epsilon)$ factor more (which, again, is just a constant overhead); and (iii) the amount of computation at the master node is $O((1+\epsilon)(1+\frac{n}{d}))$ factor more, which is comparable in cases where n is not much bigger than d.*

Remark 5. *Our scheme is not only efficient (both in terms of computational complexity and storage requirement), but it can also tolerate up to $\lfloor\frac{m-1}{2}\rfloor$ corrupt worker nodes (by taking $\epsilon = m-1$ in Theorem 1). It is not hard to prove that this bound is information-theoretically optimal, i.e., no algorithm can tolerate $\lceil\frac{m}{2}\rceil$ corrupt worker nodes, and at the same time correctly computes the gradient.*

B. Related Work

There has been significant recent interest in using coding-theoretic techniques to mitigate the well-known straggler problem [12], including gradient coding [15]–[18], encoding computation [19], [20], data encoding [13]. However, one cannot directly apply the methods for straggler mitigation to the Byzantine attacks case, as we do not know which updates are under attack. Distributed computing with Byzantine adversaries is a richly investigated topic since [6], and has received recent attention in the context of large-scale distributed optimization and learning [7]–[9], [21]. These can be divided into two categories, one which have statistical analysis/assumptions (either explicit statistical models for data [9], [21], or through stochastic GD [7]. Our method gives deterministic guarantees, distinct from these works, but similar in spirit to [8], which is the closest related work. Our storage redundancy *factor* is $2m/(m-2t)$, which is constant *even* if t is a constant fraction of m. In contrast, the storage redundancy *factor* required in [8] is $2t+1$, growing linearly with the number of corrupt worker nodes. This significantly reduces the computation time at the worker nodes in our scheme compared to the scheme in [8], without sacrificing on the computation time required by the master node. Their coding in [8] is restricted to data replication redundancy, as they encode the gradient as done in [15], enabling application to convex problems; in contrast, we encode the data enabling significantly smaller redundancy, and apply it to quadratic optimization, and is also applicable to MV multiplication. A two-round approach for gradient computation has been proposed for straggler mitigation [19], but our method for MV multiplication differs from that, as we have to provide adversarial protection. Data encoding proposed in [13] applies only to stragglers, and has low-redundancy and complexity, by allowing convergence to an approximate, rather than exact solution.

III. Our Solution

In the section, we describe the core technical part of our two round gradient computation approach – a method of performing matrix-vector (MV) multiplication in a distributed manner in the presence of a malicious adversary who can corrupt at most t of the worker nodes. Here, the matrix is fixed and we want to right-multiply a vector with this matrix.

Given a fixed matrix $\mathbf{A} \in \mathbb{R}^{n_r \times n_c}$ and a vector $\mathbf{v} \in \mathbb{R}^{n_c}$, we want to compute $\mathbf{Av}$ in a distributed manner in the presence of at most t corrupt worker nodes; see Section II for details on the model. Our method is based on data encoding and real error correction, where the matrix $\mathbf{A}$ is encoded and distributed among all the worker nodes, and the master recovers the MV product $\mathbf{Av}$ using real error correction; see Figure 1. We will think of our encoding matrix as $\mathbf{S} = [\mathbf{S}_1^T\ \mathbf{S}_2^T, \ldots, \mathbf{S}_m^T]$, where each $\mathbf{S}_i$ is a $p \times n_r$ matrix and $pm > n_r$. We will determine the value of p later and the entries of $\mathbf{S}$ later. For $i \in [m]$, we store the matrix $\mathbf{S}_i\mathbf{A}$ at the worker node i. As described in Section II, the computation proceeds as follows: The master sends $\mathbf{v}$ to all the worker nodes and receives $\mathbf{S}_i\mathbf{Av} + \mathbf{e}_i$ back from worker i, for every $i \in [m]$. Let $\mathbf{e}_i = [e_{i1}, e_{i2}, \ldots, e_{ip}]^T$ for every $i \in [p]$. Note that $\mathbf{e}_i = \mathbf{0}$ if the i'th node is honest, otherwise can be arbitrary. In order to find the corrupt worker nodes, master equivalently writes $\{\mathbf{S}_i\mathbf{Av} + \mathbf{e}_i\}_{i=1}^m$ as p systems of linear equations.

$$\tilde{h}_i(\mathbf{v}) = \tilde{\mathbf{S}}_i\mathbf{Av} + \tilde{\mathbf{e}}_i, \quad i \in [p] \tag{4}$$

where, for every $i \in [p]$, $\tilde{\mathbf{e}}_i = [e_{1i}, e_{2i}, \ldots, e_{mi}]^T$, and $\tilde{\mathbf{S}}_i$ is an $m \times n_r$ matrix whose j'th row is equal to the i'th row of $\mathbf{S}_j$, for every $j \in [m]$. Note that at most t entries in each $\tilde{\mathbf{e}}_i$ are non-zero. Observe that $\{\mathbf{S}_i\mathbf{Av} + \mathbf{e}_i\}_{i=1}^m$ and $\{\tilde{\mathbf{S}}_i\mathbf{Av} + \tilde{\mathbf{e}}_i\}_{i=1}^p$ are equivalent systems of linear equations, and we can get one from the other.

Note that $\tilde{\mathbf{S}}_i$'s constitute the encoding matrix $\mathbf{S}$, which we have to design. In the following, we will design these matrices $\tilde{\mathbf{S}}_i$'s (which in turn will determine the encoding matrix $\mathbf{S}$), with the help of another matrix $\mathbf{F}$, which will be used to find the error locations, i.e., identities of the compromised worker nodes. We will design the matrices $\mathbf{F}$ (of dimension $k \times m$, where $k < m$, which is to be determined later) and $\tilde{\mathbf{S}}_i$'s such that

- **C.1** $\mathbf{F}\tilde{\mathbf{S}}_i = 0$ for every $i \in [p]$.
- **C.2** For any t-sparse $\mathbf{u} \in \mathbb{R}^m$, we can efficiently find all the non-zero locations of $\mathbf{u}$ from $\mathbf{Fu}$.
- **C.3** For any $\mathcal{T} \subset [m]$ such that $|\mathcal{T}| \geq (m-t)$, let $\mathbf{S}_\mathcal{T}$ denote the $|\mathcal{T}|p \times n_r$ matrix obtained from $\mathbf{S}$ by restricting it to all the $\mathbf{S}_i$'s for which $i \in \mathcal{T}$. We want $\mathbf{S}_\mathcal{T}$ to be of full column rank.

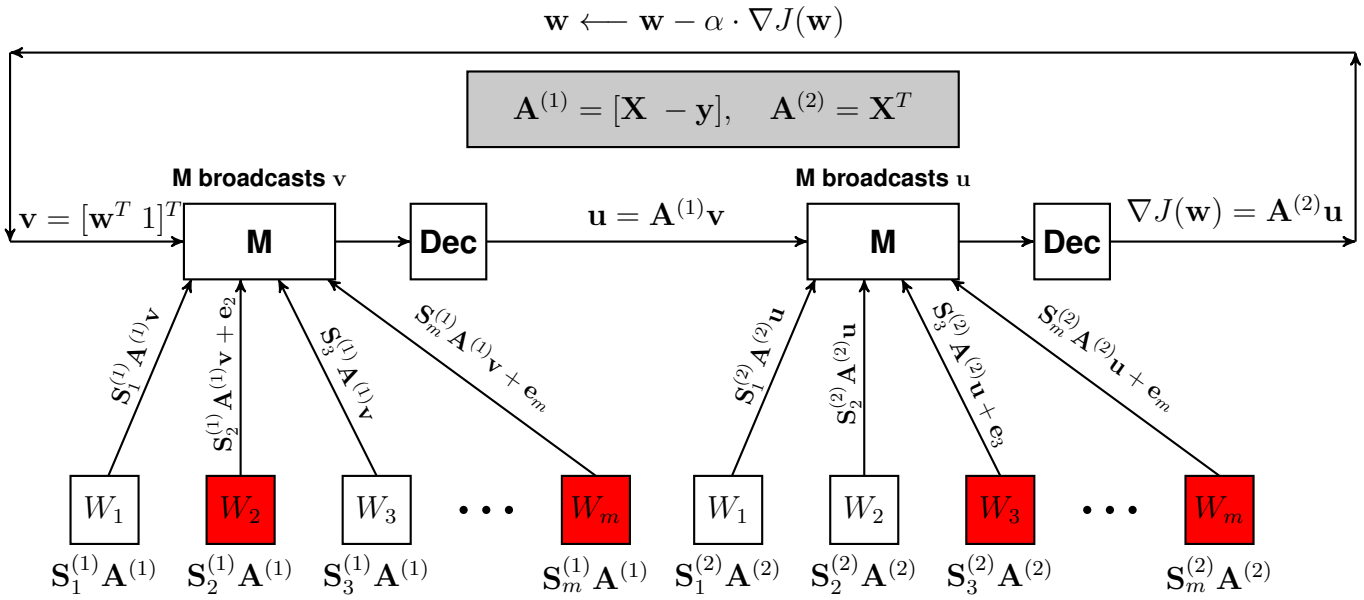

Fig. 1 This figure shows our 2-round approach to the Byzantine-resilient distributed gradient descent. Since gradient is equal to $\nabla J(\mathbf{w}) = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$, we compute it in 2 rounds, using a matrix-vector (MV) multiplication as a subroutine in each round: in the 1st round we compute $\mathbf{u} := \mathbf{A}^{(1)}\mathbf{v}$, where $\mathbf{A}^{(1)} = [\mathbf{X} \ -\mathbf{y}], \mathbf{v} = [\mathbf{w}^T \ 1]^T$, and in the second round we compute $\mathbf{X}^T\mathbf{u}$, which is equal to $\nabla J(\mathbf{w})$. To make our distributed MV multiplication $\mathbf{A}\mathbf{v}$ Byzantine-resilient, we encode $\mathbf{A}$ using a sparse matrix $\mathbf{S} = [\mathbf{S}_1^T \ \mathbf{S}_m^T \ \ldots \ \mathbf{S}_m^T]^T$ and distribute $\mathbf{S}_i\mathbf{A}$ to worker i (denoted by W_i). The adversary can corrupt at most t workers (the compromised ones are denoted in red colour), potentially different sets of t workers in different rounds. The master node (denote by **M**) broadcasts $\mathbf{v}$ to all the workers. Each worker performs the local MV product and sends it back to M. If W_i is corrupt, then it can send an arbitrary vector. Once the master has received all the vectors (out of which t may be erroneous), it sends them to the decoder (denoted by **Dec**), which outputs the correct MV product $\mathbf{A}\mathbf{v}$.

If we can find such matrices, then we can recover the desired MV multiplication $\mathbf{A}\mathbf{v}$ exactly: briefly, **C.1** and **C.2** will allow us to locate the corrupt worker nodes; once we have found them, we can discard all the information that the master node had received from them. This will yield $\mathbf{S}_{\mathcal{T}}\mathbf{A}\mathbf{v}$, where $\mathbf{S}_{\mathcal{T}}$ is the $|\mathcal{T}|p \times n_r$ matrix obtained from $\mathbf{S}$ by restricting it to $\mathbf{S}_i$'s for all $i \in \mathcal{T}$, where $\mathcal{T}$ is the set of all honest worker nodes. Now, by **C.3**, since $\mathbf{S}_{\mathcal{T}}$ is of full column rank, we can recover $\mathbf{A}\mathbf{v}$ from $\mathbf{S}_{\mathcal{T}}\mathbf{A}\mathbf{v}$ exactly. Details follow.

Suppose we have matrices $\mathbf{F}$ and $\tilde{\mathbf{S}}_i$'s such that **C.1** holds. Now, multiplying (4) by $\mathbf{F}$ yields

$$\mathbf{f}_i := \mathbf{F}\tilde{h}_i(\mathbf{v}) = \mathbf{F}\tilde{\mathbf{e}}_i, \tag{5}$$

for every $i \in [p]$, where $\|\tilde{\mathbf{e}}_i\|_0 \leq t$. In Section III-A, we give our approach for finding all the corrupt worker nodes with the help of any error locator matrix $\mathbf{F}$. Then, in Section III-B, we give a generic construction for designing $\tilde{\mathbf{S}}_i$'s (and, in turn, our encoding matrix $\mathbf{S}$) such that **C.1** and **C.3** hold. In Section III-C, we show how to compute the desired matrix-vector product $\mathbf{A}\mathbf{v}$ efficiently, once we have discarded all the data from the corrupt works nodes. Then, in Section III-D, we will give details of the error locator matrix $\mathbf{F}$ that we use in our construction.

Remark 6. *As we will see in Section III-B, the structure of our encoding matrix* $\mathbf{S}$ *is independent of our error locator matrix* $\mathbf{F}$. *Specifically, the repetitive structure of the non-zero entries of* $\mathbf{S}$ *as well as their locations will not change irrespective of what the* $\mathbf{F}$ *matrix is. This makes our construction very generic, as we can choose whichever* $\mathbf{F}$ *suits our needs the best (in terms of how many erroneous indices it can locate and with what decoding complexity), and it won't affect the structure of our encoding matrix at all – only the non-zero entries might change, neither their repetitive format, nor their locations!*

A. Finding The Corrupt Worker Nodes

Observe that $\mathsf{supp}(\tilde{\mathbf{e}}_i)$ may not be the same for all $i \in [p]$, but we know, for sure, that the non-zero locations in all these error vectors occur within the same set of t locations. Let $\mathcal{I} = \bigcup_{i=1}^{p} \mathsf{supp}(\tilde{\mathbf{e}}_i)$, which is the set of all corrupt worker nodes. Note that $|\mathcal{I}| \leq t$. We want to find this set $\mathcal{I}$ efficiently, and for that we note the following crucial observation. Since the non-zero entries of all the error vectors $\tilde{\mathbf{e}}_i$'s occur in the same set $\mathcal{I}$, a random linear combination of $\tilde{\mathbf{e}}_i$'s has support equal to $\mathcal{I}$ with probability one, if the coefficients of the linear combination are chosen from an *absolutely continuous* probability distribution. This idea has appeared before in [22] in the context of compressed sensing for recovering arbitrary sets of jointly sparse signals that have been measured by the same measurement matrix.

Definition 1. *A probability distribution is called* absolutely continuous*, if every event of measure zero occurs with probability zero.*

It is well-known that a distribution is absolutely continuous if and only if it can be represented as an integral over an integrable density function [23, Theorem 31.8, Chapter 6]. Since Gaussian and uniform distributions have an explicit integrable

density function, both are absolutely continuous. Conversely, discrete distributions are not absolutely continuous. Now we state a lemma from [22] that shows that a random linear combination of the error vectors (where coefficients are chosen from an absolutely continuous distribution) preserves the support with probability one.

Lemma 1 ([22]). *Let $\mathcal{I} = \bigcup_{i=1}^{p} \mathsf{supp}(\tilde{\mathbf{e}}_i)$, and let $\hat{\mathbf{e}} = \sum_{i=1}^{p} \alpha_i \tilde{\mathbf{e}}_i$, where α_i's are sampled i.i.d. from an absolutely continuous distribution. Then with probability 1, $\mathsf{supp}(\hat{\mathbf{e}}) = \mathcal{I}$.*

From (5) we have $\mathbf{f}_i = \mathbf{F}\tilde{\mathbf{e}}_i$ for every $i \in [p]$. Take a random linear combination of $\mathbf{f}_i$'s with coefficients α_i's chosen i.i.d. from an absolutely continuous distribution, for example, the Gaussian distribution. Let $\tilde{\mathbf{f}} = \alpha_i \left(\sum_{i=1}^{p} \mathbf{f}_i\right) = \alpha_i \left(\sum_{i=1}^{p} \mathbf{F}\tilde{\mathbf{e}}_i\right) = \mathbf{F}\left(\sum_{i=1}^{p} \alpha_i \tilde{\mathbf{e}}_i\right) = \mathbf{F}\tilde{\mathbf{e}}$, where $\tilde{\mathbf{e}} = \sum_{i=1}^{p} \alpha_i \tilde{\mathbf{e}}_i$. Note that, with probability 1, $\mathsf{supp}(\tilde{\mathbf{e}})$ is equal to the set of all corrupt worker nodes, and we want to find this set efficiently. In other words, given $\mathbf{F}\tilde{\mathbf{e}}$, we want to find $\mathsf{supp}(\tilde{\mathbf{e}})$ efficiently. For this, we need to design a $k \times m$ matrix $\mathbf{F}$ (where $k < m$) such that for any sparse error vector $\mathbf{e} \in \mathbb{R}^m$, we can efficiently find $\mathsf{supp}(\mathbf{e})$. Many such matrices have been known in the literature that can handle different levels of sparsity with varying decoding complexity. We can choose any of these matrices depending on our need, and this will not affect the design of our encoding matrix $\mathbf{S}$. In particular, we will use a $k \times m$ Vandermonde matrix along with the Reed-Solomon decoding, which can correct up to $k/2$ errors and has decoding complexity of $O(m^2)$; see Section III-D for details.

Time required in finding the corrupt worker nodes. The time taken in finding the corrupt worker nodes is equal to the sum of the time taken in the following 3 tasks. (i) Computing $\mathbf{F}\tilde{\mathbf{e}}_i$ for every $i \in [p]$: Note that we can get $\mathbf{F}\tilde{\mathbf{e}}_i$ by multiplying (4) with $\mathbf{F}$. Since $\mathbf{F}$ is a $k \times m$ matrix, and we compute $\mathbf{F}\tilde{h}_i(\mathbf{v})$ for p systems, this requires $O(pkm)$ time. (ii) Taking a random linear combination of p vectors each of length m, which takes $O(pm)$ time. (iii) Applying Lemma 2 (in Section III-D) once to find the error locations, which takes $O(m^2)$ time. Since p is much bigger than m, the total time complexity is $O(pkm)$.

B. Designing The Encoding Matrix $\mathbf{S}$

Now we give a generic construction for designing $\tilde{\mathbf{S}}_i$'s such that **C.1** and **C.3** hold. Fix any $k \times m$ matrix $\mathbf{F}$ such that we can efficiently find $\mathbf{e}$ from $\mathbf{Fe}$, provided $\mathbf{e}$ is sufficiently sparse. We can assume, without loss of generality, that $\mathbf{F}$ has full row-rank; otherwise, there will be redundant observations in $\mathbf{Fe}$ that we can discard and make $\mathbf{F}$ smaller by discarding the redundant row. Let $\mathcal{N}(\mathbf{F}) \subset \mathbb{R}^m$ denote the null-space of $\mathbf{F}$. Since $\mathsf{rank}(\mathbf{F}) = k$, dimension of $\mathcal{N}(\mathbf{F})$ is $q = (m - k)$. Let $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_q\}$ be a basis of $\mathcal{N}(\mathbf{F})$, and let $\mathbf{b}_i = [b_{i1}\ b_{i2} \ldots b_{im}]^T$, for every $i \in [q]$. We set $\mathbf{b}_i$'s the columns of the following matrix $\mathbf{F}^{\perp}$:

$$\mathbf{F}^{\perp} = \begin{bmatrix} b_{11} & b_{21} & \ldots & b_{q1} \\ b_{12} & b_{22} & \ldots & b_{q2} \\ \vdots & \vdots & \vdots & \vdots \\ b_{1m} & b_{2m} & \ldots & b_{qm} \end{bmatrix}_{m \times q} \tag{6}$$

The following property of $\mathbf{F}^{\perp}$ will be used for recovering the MV product in Section III-C.

Claim 1. *For any subset $\mathcal{T} \subset [m]$, such that $|\mathcal{T}| \geq (m - t)$, let $\mathbf{F}^{\perp}_{\mathcal{T}}$ be the $|\mathcal{T}| \times q$ matrix, which is equal to the restriction of $\mathbf{F}^{\perp}$ to the rows in $\mathcal{T}$. Then $\mathbf{F}^{\perp}_{\mathcal{T}}$ is of full column rank.*

Proof. Note that $q = m - k$, where $k = 2t$. So, if we show that any q rows of $\mathbf{F}^{\perp}$ are linearly independent, then, this in turn will imply that for every $\mathcal{T} \subset [m]$ with $|\mathcal{T}| \geq (m - t)$, the sub-matrix $\mathbf{F}^{\perp}_{\mathcal{T}}$ will have full column rank. In the following we show that any q rows of $\mathbf{F}^{\perp}$ are linearly independent. To the contrary, suppose not; and let $\mathcal{T}' \subset [m]$ with $|\mathcal{T}'| = q$ be such that the $q \times q$ matrix $\mathbf{F}^{\perp}_{\mathcal{T}'}$ is not a full rank matrix. This implies that there exists a non-zero $\mathbf{c}' \in \mathbb{R}^q$ such that $\mathbf{F}^{\perp}_{\mathcal{T}'}\mathbf{c}' = \mathbf{0}$. Let $\mathbf{b} = \mathbf{F}^{\perp}\mathbf{c}'$. Note that $\mathbf{b} \neq \mathbf{0}$ (because columns of $\mathbf{F}^{\perp}$ are linearly independent) and also that $\|\mathbf{b}\|_0 \leq m - q = k$. Now, since $\mathbf{F}\mathbf{F}^{\perp} = \mathbf{0}$, we have $\mathbf{F}\mathbf{b} = \mathbf{0}$, which contradicts the fact that any k columns of $\mathbf{F}$ are linearly independent. □

Now we design $\tilde{\mathbf{S}}_i$'s. For $i \in [p]$, we set $\tilde{\mathbf{S}}_i$ as follows:

$$\tilde{\mathbf{S}}_i = \begin{bmatrix} 0 & \ldots & 0 & b_{11} & b_{21} & \ldots & b_{l1} & 0 & \ldots & 0 \\ 0 & \ldots & 0 & b_{12} & b_{22} & \ldots & b_{l2} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & b_{1m} & b_{2m} & \ldots & b_{lm} & 0 & \ldots & 0 \end{bmatrix}$$

where $l = q$ if $i < p$; otherwise $l = n_r - (p-1)q$. The first $(i-1)q$ and the last $n_r - [(i-1)q + l]$ columns of $\tilde{\mathbf{S}}_i$ are zero. This also implies that the number of rows in each $\mathbf{S}_i$ is $p = \lceil n_r/q \rceil$.

Claim 2. *For every $i \in [p]$, we have $\mathbf{F}\tilde{\mathbf{S}}_i = 0$.*

Proof. By construction, the null-space of $\mathbf{F}$ is $\mathcal{N}(\mathbf{F}) = \mathsf{span}\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_q\}$, which implies that $\mathbf{F}\mathbf{b}_i = \mathbf{0}$, for every $i \in [q]$. Since all the columns of $\tilde{\mathbf{S}}_i$'s are either $\mathbf{0}$ or $\mathbf{b}_j$ for some $j \in [q]$, the claim follows. □

The above constructed matrices $\tilde{\mathbf{S}}_i$'s give the following encoding matrix $\mathbf{S}_i$ for the i'th worker node:

$$\mathbf{S}_i = \begin{bmatrix} b_{1i} \ldots b_{qi} & & & \\ & \ddots & & \\ & & b_{1i} \ldots b_{qi} & \\ & & & b_{1i} \ldots b_{li} \end{bmatrix}_{p \times n_r} \tag{7}$$

All the unspecified entries of $\mathbf{S}_i$ are zero. The matrix $\mathbf{S}_i$ is for encoding the data for worker i. By stacking up the $\mathbf{S}_i$'s horizontally gives us our desired encoding matrix $\mathbf{S}$.

To get efficient encoding, we want $\mathbf{S}$ to be as sparse as possible. Since $\mathbf{S}$ is completely determined by $\mathbf{F}^{\perp}$, whose columns are the basis vectors of $\mathcal{N}(\mathbf{F})$, it suffices to find a sparse basis for $\mathcal{N}(\mathbf{F})$. It is known that finding the sparsest basis for the null-space of a matrix is NP-hard [24]. Note that we can always find the basis vectors of $\mathcal{N}(\mathbf{F})$ by reducing $\mathbf{F}$ to its row-reduced-echelon-form (RREF) using the Gaussian elimination [25]. This will result in $\mathbf{F}^{\perp}$ whose last q rows forms a $q \times q$ identity matrix. Note that $q = m - k$, where $k = 2t$. So, if the corruption threshold t is very small as compared to m, the $\mathbf{F}^{\perp}$ that we obtain by the RREF will be very sparse – only the first $2t$ rows may be dense. Since computing $\mathbf{S}$ is equivalent to computing $\mathbf{F}^{\perp}$, and we can compute $\mathbf{F}^{\perp}$ in $O(k^2 m)$ time using the Gaussian elimination, the time complexity of computing $\mathbf{S}$ is also $O(k^2 m)$.

Now we prove an important property of the encoding matrix $\mathbf{S}$ that will be crucial for recovery of the desired matrix-vector product.

Claim 3. *For any $\mathcal{T} \subset [m]$ such that $|\mathcal{T}| \geq (m-t)$, let $\mathbf{S}_{\mathcal{T}}$ denote the $|\mathcal{T}|p \times n_r$ matrix obtained from $\mathbf{S}$ by restricting it to all the blocks $\mathbf{S}_i$'s for which $i \in \mathcal{T}$. Then $\mathbf{S}_{\mathcal{T}}$ is of full column rank.*

Proof. For $i \in [p-1]$, let $\mathcal{B}_i = [(i-1)q + 1 : iq]$ and $\mathcal{B}_p = [(p-1)q + 1 : n_r - (p-1)q]$, where we see $\mathcal{B}_i$'s as a collection of some column indices. Consider any two distinct $i, j \in [p]$. It is clear that for any two vectors $\mathbf{u}_1 \in \mathcal{B}_i, \mathbf{u}_2 \in \mathcal{B}_j$, we have $\mathsf{supp}(\mathbf{u}_1) \cap \mathsf{supp}(\mathbf{u}_2) = \phi$, which means that all the columns in distinct $\mathcal{B}_i$'s are linearly independent. So, to prove the claim, we only need to show that the columns within the same $\mathcal{B}_i$'s are linearly independent. Fix any $i \in [p]$, and consider the $|\mathcal{T}|p \times q$ sub-matrix $\mathbf{S}_{\mathcal{T}}^{(i)}$ of $\mathbf{S}_{\mathcal{T}}$, which is obtained by restricting $\mathbf{S}_{\mathcal{T}}$ to the columns in $\mathcal{B}_i$. There are precisely $|\mathcal{T}|$ non-zero rows in $\mathbf{S}_{\mathcal{T}}^{(i)}$, which are equal to the rows of the matrix $\mathbf{F}_{\mathcal{T}}^{\perp}$ defined in Claim 1. We have already shown in the proof of Claim 1 that $\mathbf{F}_{\mathcal{T}}^{\perp}$ is of full column rank. Therefore, $\mathbf{S}_{\mathcal{T}}^{(i)}$ is also of full column rank. This concludes the proof of Claim 3. □

Since $\mathbf{S}_{\mathcal{T}}$ is of full column rank, in principle, we can recover any vector $\mathbf{u} \in \mathbb{R}^{n_r}$ from $\mathbf{S}_{\mathcal{T}}\mathbf{u}$. In the next section, we show an efficient way for this recovery.

C. Recovering The Matrix-Vector Product $\mathbf{Av}$

Once the master has found the set $\mathcal{I}$ of corrupt worker nodes, it discards all the data received from them. Let $\mathcal{T} = [m] \setminus \mathcal{I} = \{i_1, i_2, \ldots, i_f\}$ be the set of all honest worker nodes, where $f = (m - |\mathcal{I}|) \geq (m - t)$. Let $\mathbf{r} = [\mathbf{r}_1^T \mathbf{r}_2^T \ldots \mathbf{r}_m^T]$, where $\mathbf{r}_i = \mathbf{S}_i\mathbf{Av} + \mathbf{e}_i$. All the $\mathbf{r}_i$'s from the honest worker nodes can be written as

$$\mathbf{r}_{\mathcal{T}} = \mathbf{S}_{\mathcal{T}}\mathbf{Av}, \tag{8}$$

where $\mathbf{S}_{\mathcal{T}}$ is as defined in Claim 3, and $\mathbf{r}_{\mathcal{T}}$ is also defined analogously and equal to the restriction of $\mathbf{r}$ to all the $\mathbf{r}_i$'s for which $i \in \mathcal{T}$. Since $\mathbf{S}_{\mathcal{T}}$ has full column rank (by Claim 3), in principle, we can recover $\mathbf{Av}$ from (8). Next we show how to recover $\mathbf{Av}$ efficiently, by exploiting the structure of $\mathbf{S}$.

Let $\tilde{\mathbf{r}}_j = [r_{i_1 j}, r_{i_2 j}, \ldots, r_{i_f j}]^T$, for every $j \in [p]$. The repetitive structure of $\mathbf{S}_i$'s (see (7)) allows us to write (8) equivalently in terms of p smaller systems.

$$\tilde{\mathbf{r}}_j = \mathbf{F}_j(\mathbf{Av})_{\mathcal{B}_j}, \quad \text{for } j \in [p], \tag{9}$$

where, for $j \in [p-1]$, $\mathcal{B}_i = [(i-1)q+1 : iq]$ and $\mathbf{F}_j = \mathbf{F}_{\mathcal{T}}^{\perp}$, and $\mathcal{B}_p = [(p-1)q+1 : n_r - (p-1)q]$ and $\mathbf{F}_p$ is equal to the restriction of $\mathbf{F}_{\mathcal{T}}^{\perp}$ to its first $(n_r - (p-1)q)$ columns. Since $\mathbf{F}_{\mathcal{T}}^{\perp}$ has full column rank (by Claim 1), we can compute $(\mathbf{Av})_{\mathcal{B}_i}$ for all $i \in [p]$, by multiplying (9) by $\mathbf{F}_j^{+} = (\mathbf{F}_j^T\mathbf{F}_j)^{-1}\mathbf{F}_j^T$, which it called the Moore-Penrose inverse of $\mathbf{F}_j$. Since $\mathbf{Av} = [(\mathbf{Av})_{\mathcal{B}_1}^T, (\mathbf{Av})_{\mathcal{B}_2}^T, \ldots, (\mathbf{Av})_{\mathcal{B}_p}^T]^T$, we can recover the desired MV product $\mathbf{Av}$.

Time Complexity analysis. The task of obtaining $\mathbf{Av}$ from $\mathbf{S}_{\mathcal{T}}\mathbf{Av}$ reduces to (i) computing $\mathbf{F}_j^{+} = (\mathbf{F}_{\mathcal{T}}^{\perp})^{+}$ once, which takes $O(q^2|\mathcal{T}|)$ time naïvely; (ii) computing $\mathbf{F}_p^{+}$ once, which takes at most $O(q^2|\mathcal{T}|)$ time naïvely; and (iii) computing the MV products $\mathbf{F}_j^{+}\tilde{\mathbf{r}}_j$ for every $j \in [p]$, which takes $O(pq|\mathcal{T}|)$ time in total. Since p is much bigger than q, the total time taken in recovering $\mathbf{Av}$ from $\mathbf{S}_{\mathcal{T}}\mathbf{Av}$ is $O(pq|\mathcal{T}|) = O(pm^2)$.

D. Designing The Error Locator Matrix $\mathbf{F}$

In this section, we design a $k \times m$ matrix $\mathbf{F}$ (where $k < m$) such that for any sparse error vector $\mathbf{e} \in \mathbb{R}^m$, we can efficiently find $\mathsf{supp}(\mathbf{e})$. Many such matrices have been known in the literature (for recovering the vector $\mathbf{e}$ given $\mathbf{Fe}$) since the work of [10], that can handle different levels of sparsity with varying decoding complexity. Most of these are random constructions, which may not work with small block-lengths (in our setting, m is a small number). Furthermore, they can only correct a constant fraction of errors, where the constant is very small. We need a deterministic construction that can handle a constant fraction (ideally up to 1/2) of errors and that works with small block-lengths.

Akçakaya and Tarokh [26] constructed a complex analogue of the Reed-Solomon codes from $k \times m$ Vandermonde matrices $\mathbf{F}$ and gave an $O(m^2)$ time algorithm for exactly reconstructing $\mathbf{e}$ from $\mathbf{f} = \mathbf{Fe}$, provided $|\mathsf{supp}(\mathbf{e})| \leq k/2$. Let $z_1, z_2, \ldots, z_m$ be m distinct non-zero elements in $\mathbb{R}$. We define $\mathbf{F}$ to be the following Vandermonde matrix (where $k < m$):

$$\mathbf{F} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ z_1 & z_2 & z_3 & \ldots & z_m \\ z_1^2 & z_2^2 & z_3^2 & \ldots & z_m^2 \\ \vdots & \vdots & \ldots & \ldots & \vdots \\ z_1^{k-1} & z_2^{k-1} & z_3^{k-1} & \ldots & z_m^{k-1} \end{bmatrix}_{k \times m} \tag{10}$$

Below we state a result (specialized to reals) from [26].

Lemma 2 ([26])**.** *Let $\mathbf{F}$ be the $k \times m$ matrix as defined in* (10)*. Let $\mathbf{e} \in \mathbb{R}^m$ be an arbitrary vector with $|\mathsf{supp}(\mathbf{e})| \leq k/2$. We can exactly recover the vector $\mathbf{e}$ from $\mathbf{f} = \mathbf{Fe}$ in $O(m^2)$ time.*

Note that $\mathbf{F}$ is a $k \times m$ matrix, where $k < m$. Choosing k is in our hands, and larger the k, more the number of errors

we can correct (but at the expense of increased storage and computation); see Section IV for more details.

IV. Resource Requirement Analysis

In this section, we analyze the total amount of resources (storage and computation) required by our method that we developed in this paper for the Byzantine-resilient distributed gradient descent (GD) in the presence of t adversarial worker nodes (which may be different in different rounds) and prove Theorem 1. Fix an $\epsilon > 0$. Let the corruption threshold t satisfies $t \leq \lfloor(\epsilon/(1+\epsilon)) \cdot (m/2)\rfloor$.

As described earlier in Section II-A, we compute the gradient $\nabla J(\mathbf{w}) = \mathbf{X}^T(\mathbf{X}\mathbf{w}-\mathbf{y})$ in two-rounds; and in each round we use the Byzantine-tolerant MV multiplication, which we have developed in Section III, as a subroutine; see Figure 1 for a pictorial representation of our scheme. To compute $\mathbf{X}\mathbf{w} - \mathbf{y}$ in the first round, we encode $\mathbf{A}^{(1)} = [\mathbf{X} \ \ -\mathbf{y}]$ and compute $\mathbf{u} = \mathbf{A}^{(1)}\mathbf{v}$, where $\mathbf{v} = [\mathbf{w}^T \ 1]^T$. To compute $\mathbf{X}^T\mathbf{u}$ (which is equal to the gradient) in the second round, we encode $\mathbf{A}^{(2)} = \mathbf{X}^T$ and compute $\mathbf{A}^{(2)}\mathbf{u}$. Let $\mathbf{S}^{(1)}$ and $\mathbf{S}^{(2)}$ be the encoding matrices of dimensions $p_1 m \times n$ and $p_2 m \times d$, respectively, to encode $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$, respectively. Here, $p_1 = \lceil n/q \rceil$ and $p_2 = \lceil d/q \rceil$, where $q = m - k$. Since $k = 2t$ (by Lemma 2), we have $q = (m-k) \geq m/(1+\epsilon)$.

A. Storage Requirement

Each worker node i stores two matrices $\mathbf{S}_i^{(1)}\mathbf{A}^{(1)}$ and $\mathbf{S}_i^{(2)}\mathbf{A}^{(2)}$. The first one is a $p_1 \times (d+1)$ matrix, and the second one is a $p_2 \times n$ matrix. So, the total amount of storage at all worker nodes is equal to storing $(p_1(d+1)+p_2 n) \times m$ real numbers. Since $p_1 \leq \lceil (1+\epsilon)\frac{n}{m} \rceil$ and $p_2 \leq \lceil (1+\epsilon)\frac{d}{m} \rceil$, the total storage is

$$\begin{aligned}\big(p_1(d+1)+p_2 n\big)m &= p_1 m(d+1) + p_2 mn \\ &< [(1+\epsilon)n + m](d+1) + [(1+\epsilon)d + m]n \\ &= (1+\epsilon)n(2d+1) + m(n+d+1).\end{aligned}$$

where the first term is roughly equal to a $2(1+\epsilon)$ factor more than the size of $\mathbf{X}$. Note that the second term does not contribute much to the total storage as compared to the first term, because the number of worker nodes m is much smaller than both n and d. In fact, if $m-k$ divides both n and d, then the second term vanishes. Since $|\mathbf{X}|$ is an $n \times d$ matrix, the total storage at each worker node is almost equal to $2(1+\epsilon)\frac{|\mathbf{X}|}{m}$, which is a constant factor of the optimal, that is, $\frac{|\mathbf{X}|}{m}$, and the total storage is roughly equal to $2(1+\epsilon)|\mathbf{X}|$.

B. Computational Complexity

We can divide the computational complexity of our scheme as follows:

1) Encoding the data matrix: Since, for every $i \leq k$ and $j > k$, the total number of non-zero entries in $\mathbf{S}_i^{(1)}$ and $\mathbf{S}_j^{(1)}$ are at most n and p_1, respectively (see Section III-B for details), the computational complexity for computing $\mathbf{S}_i^{(1)}\mathbf{A}$ for each $i \leq k$, and $\mathbf{S}_j^{(1)}\mathbf{A}$ for each $j > k$, is $O(nd)$ and $O(p_1 d)$, respectively. So, the encoding time for computing $\mathbf{S}^{(1)}\mathbf{A}^{(1)}$ is equal to $O\left(k(nd) + (m-k)(pd)\right) = O\left((\frac{\epsilon}{1+\epsilon}m + 1)nd\right)$. Similarly, we can show that the encoding time for computing $\mathbf{S}^{(2)}\mathbf{A}^{(2)}$ is also equal to $O\left((\frac{\epsilon}{1+\epsilon}m+1)nd\right)$. Note that computing $\mathbf{S}^{(1)}$ and $\mathbf{S}^{(2)}$ take $O(k^2 m)$ time each, which is much smaller compared to the encoding time. So, the total encoding time is $O\left((\frac{\epsilon}{1+\epsilon}m+1)nd\right)$. Note that this encoding is to be done only once.

2) Computation at each worker node: In the first round, upon receiving $\mathbf{v}$ from the master node, each worker i computes $(\mathbf{S}_i^{(1)}\mathbf{A}^{(1)})\mathbf{v}$, and reports back the resulting vector. Similarly, in the second round, upon receiving $\mathbf{u}$ from the master node, each worker i computes $(\mathbf{S}_i^{(2)}\mathbf{A}^{(2)})\mathbf{u}$, and reports back the resulting vector. Since $\mathbf{S}_i^{(1)}\mathbf{A}^{(1)}$ and $\mathbf{S}_i^{(2)}\mathbf{A}^{(2)}$ are $p_1 \times (d+1)$ and $p_2 \times n$ matrices, respectively, each worker node i requires $O(p_1 d + p_2 n) = O((1+\epsilon)\frac{nd}{m})$ time.

3) Computation at the master node: The total time taken by the master node in both the rounds is the sum of the time required in (i) finding the corrupt worker nodes in the 1st and 2nd rounds, which requires $O(p_1 km)$ and $O(p_2 km)$ time, respectively (see Section III-A), and (ii) recovering $\mathbf{A}^{(1)}\mathbf{v}$ from $\mathbf{S}_{\mathcal{T}}^{(1)}\mathbf{A}^{(1)}\mathbf{v}$ in the 1st round, which requires $O(p_1 m^2)$ time, and recovering $\mathbf{A}^{(2)}\mathbf{v}$ from $\mathbf{S}_{\mathcal{T}}^{(2)}\mathbf{A}^{(2)}\mathbf{u}$ in the 2nd round, which requires $O(p_2 m^2)$ time (see Section III-C). Since $k < m$, the total time is equal to $O((p_1+p_2)m^2) = O((1+\epsilon)(n+d)m)$.

V. Encoding In The Streaming Model

An attractive property of our encoding scheme is that it is very easy to update with new data points. More specifically, our encoding requires the same amount of time, irrespective of whether we get all the data at once, or we get each sample point one by one, as in the online/streaming model. This setting encompasses a more realistic scenario, in which we design our coding scheme with the initial set of data points and distribute the encoded data among the workers. Later on, when we get some more samples, we can easily incorporate them into our existing encoded data. The update with a new sample point $\mathbf{x} \in \mathbb{R}^d$ requires $T = O\left((\frac{\epsilon}{1+\epsilon}m+1)d\right)$ time in total. This is optimal, since the offline encoding of n data points requires nT time. At the end of the update, the final encoded matrix that we get is the same as the one we would have got had we had all the $n+1$ data points in the beginning. Therefore, the decoding is not affected by this method at all. The efficient update property of our coding scheme is made possible by the repetitive structure of our encoding matrix; see (7). This structure is independent of the number of data points n and the dimension d; it only depends on the number of worker nodes and the corruption threshold. We remark that other data encoding methods in literature, even for weaker models, do not support efficient update; for example, the encoding of [13], which was designed for mitigating stragglers, depends on the dimensions n and d of the data matrix. So, it may not efficiently update if a new data point comes in.

VI. Conclusion

In this paper, we focused on the squared loss function and proposed a solution to the Byzantine-resilient distributed gradient descent (GD) algorithm in the master-worker architecture, where a malicious adversary can corrupt up to t of the m worker machines, and the compromised ones can arbitrarily deviate from their pre-specified programs. Our solution is generic and efficient, and is based on data encoding using sparse encoding matrices and real error correction for decoding. In our coding scheme, we distributed the encoded data matrix among the workers and compute the gradient at the master node using real error correction. There is a trade-off between the corruption threshold and the computational complexity & the storage required by our coding scheme. So, depending on the scenario, we can choose the parameters that work best. In particular, with constant overhead on the computational complexity and the storage requirement as compared to running the distributed GD *without any adversarial protection*, our scheme can tolerate up to 1/3 of the corrupt worker nodes. We can tolerate up to $\lfloor \frac{m-1}{2} \rfloor$ corrupt worker nodes. Since our encoding matrix is sparse, the encoding time complexity only incurs a factor of $(2t+1)$ more than what is required by the plain distributed GD algorithm without an adversary just for distributing the raw data matrix among m worker nodes. In addition, because of the repetitive structure of the encoding matrix, encoding of data in the streaming model is as efficient as encoding when all the data is available offline.

Acknowledgements

The work of the authors was partially supported by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196, by the UC-NL grant LFR-18-548554, and by the NSF award 1740047. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[3] X. Lian, C. Zhang, H. Zhang, C. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems, NIPS 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 5336–5346.

[4] T. F. Abdelzaher *et al.*, "Will distributed computing revolutionize peace? the emergence of battlefield iot," in *ICDCS 2018*, 2018, pp. 1129–1138.

[5] J. Konecný, "Stochastic, distributed and federated optimization for machine learning," Ph.D. dissertation, University of Edinburgh, 2017.

[6] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[7] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems, NIPS 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 118–128.

[8] L. Chen, H. Wang, Z. B. Charles, and D. S. Papailiopoulos, "DRACO: byzantine-resilient distributed training via redundant gradients," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 902–911.

[9] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *POMACS*, vol. 1, no. 2, pp. 44:1–44:25, 2017.

[10] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[12] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[13] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *In Advances in Neural Information Processing Systems, NIPS 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 5440–5448.

[14] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 3518–3527.

[15] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 3368–3376.

[16] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic MDS codes and expander graphs," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 4302–4310.

[17] Z. B. Charles and D. S. Papailiopoulos, "Gradient coding using the stochastic block model," in *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, 2018, pp. 1998–2002.

[18] W. Halbawi, N. A. Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, 2018, pp. 2027–2031.

[19] K. Lee, M. Lam, R. Pedarsani, D. S. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[20] S. Dutta, V. R. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 2092–2100.

[21] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 5636–5645.

[22] M. Mishali and Y. C. Eldar, "Reduce and boost: Recovering arbitrary sets of jointly sparse vectors," *IEEE Transactions on Signal Processing*, vol. 56, no. 10, pp. 4692–4702, Oct 2008.

[23] P. Billingsley, *Probability and Measure*, ser. Wiley Series in Probability and Statistics. Wiley, 1995.

[24] T. F. Coleman and A. Pothen, "The null space problem I. complexity," *SIAM Journal on Algebraic Discrete Methods*, vol. 7, no. 4, pp. 527–537, 1986.

[25] K. M. Hoffman and R. Kunze, *Linear algebra*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[26] M. Akçakaya and V. Tarokh, "A frame construction and a universal distortion bound for sparse representations," *IEEE Trans. Signal Processing*, vol. 56, no. 6, pp. 2443–2450, 2008.